

## 1) Push and Pop Operations

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  struct Stack{
5      int size;
6      int top;
7      int *arr;
8  };
9
10 void push(struct Stack *s,int element){
11     if(s->top==s->size-1){
12         printf("Stack Overflow\n");
13     }
14     else{
15         s->top++;
16         s->arr[s->top]=element;
17         printf("%d inserted\n",element);
18     }
19 }
20 int pop(struct Stack *s){
21     int t;
22     if(s->top== -1){
23         printf("Stack Underflow\n");
24     }
25     else{
26         t=s->arr[s->top];
27         s->top--;
28         printf("%d popped out\n",t);
29         return t;
30     }
31 }
32 int main(){
33     struct Stack s;
34     printf("Enter the length of array : ");
35     scanf("%d",&s.size);
36     s.arr=(int*)malloc(s.size*sizeof(int));
37     s.top=-1;
38     int choice,ele;
39     bool again= true;
```

```
39     bool again= true;
40     while(again){
41         printf("Enter 1 to push\n");
42         printf("Enter 2 to pop\n");
43         printf("Enter 0 to stop\n");
44         scanf("%d",&choice);
45         switch (choice)
46         {
47             case 1:
48                 printf("Enter a element : ");
49                 scanf("%d",&ele);
50                 push(&s,ele);
51                 break;
52             case 2:
53                 pop(&s);
54                 break;
55             case 0:
56                 again=false;
57                 break;
58             default:
59                 break;
60         }
61     }
62     return 0;
63 }
```

```
PS D:\ENGINEERING\DSA_C\PRAC_2> cd "d:\ENGINEERING\DSA_C\stack"
Enter the length of array : 2
Enter 1 to push
Enter 2 to pop
Enter 0 to stop
1
Enter a element : 5
5 inserted
Enter 1 to push
Enter 2 to pop
Enter 0 to stop
1
Enter a element : 8
8 inserted
Enter 1 to push
Enter 2 to pop
Enter 0 to stop
1
Enter a element : 6
Stack Overflow
Enter 1 to push
Enter 2 to pop
Enter 0 to stop
2
8 popped out
Enter 1 to push
Enter 2 to pop
Enter 0 to stop
2
5 popped out
Enter 1 to push
Enter 2 to pop
Enter 0 to stop
2
Stack Underflow
Enter 1 to push
Enter 2 to pop
Enter 0 to stop
0
PS D:\ENGINEERING\DSA_C\stack> |
```

## 2)Infix to Postfix and Solving postfix Expression

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  struct Stack {
5      int size;
6      int top;
7      char *arr;
8  };
9  int stackTop(struct Stack *sp){
10     return sp->arr[sp->top];
11 }
12
13 int isEmpty(struct Stack *s){
14     if(s->top== -1){
15         return 1;
16     }
17     return 0;
18 }
19 int isFull(struct Stack *s){
20     if(s->top==s->size-1){
21         return 1;
22     }
23     return 0;
24 }
25
26 void push(struct Stack *s,int element){
27     if(isFull(s)){
28         printf("Stack Overflow");
29     }
30     else{
31         s->top++;
32         s->arr[s->top]=element;
33     }
34 }
35
36 char pop(struct Stack *s){
37     char t;
38     if(isEmpty(s)){
39         printf("Stack Underflow");
40     }
41     else{
42         t=s->arr[s->top];
43         s->top--;
44         return t;
45     }
46 }
47
48 int precedence(char ch){
49     if(ch=='*' || ch=='/'){
50         return 3;
51     }
52     else if(ch=='+' || ch=='-'){
53         return 2;
54     }
55     return 0;
56 }
57
58 int isOperator(char ch){
59     if(ch=='+' || ch=='-' || ch=='*' || ch=='/'){
60         return 1;
61     }
62 }
63     return 0;
64 }
```

```

66 char *infixToPostfix(char *infix){
67     struct Stack *sp=(struct Stack *)malloc(sizeof(struct Stack)) ;
68     sp->size=100;
69     sp->top=-1;
70     sp->arr=(char*)malloc(sp->size*sizeof(char));
71     char *postfix=(char*)malloc((strlen(infix)+1)*sizeof(char));
72     int i=0,j=0;
73     while(infix[i]!='\0'){
74         if(!isOperator(infix[i])){
75             postfix[j]=infix[i];
76             i++;j++;
77         }
78         else{
79             if(precedence(infix[i])>precedence(stackTop(sp))){
80                 push(sp,infix[i]);
81                 i++;
82             }
83             else{
84                 postfix[j]=pop(sp);
85                 j++;
86             }
87         }
88     }
89     while(!isEmpty(sp)){
90         postfix[j]=pop(sp);
91         j++;
92     }
93     postfix[j]='\0';
94     return postfix;
95 }
96 int solve(char *eqn){
97     int len=strlen(eqn);
98     struct Stack *sp;
99     sp->top=-1;
100    sp->arr=(char*)malloc(sizeof(char)*11);
101    for (int i = 0; eqn[i]!='\0'; i++)
102    {
103        if(isdigit(eqn[i])){
104            push(sp,eqn[i]-48);
105        }
106        else if(isOperator(eqn[i])==1){
107            int var2=pop(sp);
108            int var1=pop(sp);
109            switch (eqn[i])
110            {
111                case '+':
112                    push(sp,var1+var2);
113                    break;
114                case '-':
115                    push(sp,var1-var2);
116                    break;
117                case '/':
118                    push(sp,var1/var2);
119                    break;
120                case '*':
121                    push(sp,var1*var2);
122                    break;
123                default:
124                    break;
125            }
126        }
127    }
128    return pop(sp);
129 }
130
131 int main()
132 {
133     char *infix="8+8/4+3";
134     char *postfix=infixToPostfix(infix);
135     printf("Postfix eqn : %s\n",postfix);
136     printf("Ans : %d",solve(postfix));
137     return 0;}

```

```

PS D:\ENGINEERING\DSA_C\
Infix eqn : 8+8/4+3
Postfix eqn : 884/+3+
Ans : 13

```

